2

RL-TR-93-158
In-House Report
September 1993

STATIC TASK ALLOCATION FOR PARALLEL PROCESSING SYSTEMS DURING SOFTWARE DEVELOPMENT

Richard C. Metzger, John K. Antonio, Loretta S. Auvil

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-93-158 has been reviewed and is approved for publication.

APPROVED: Samuel A DiNitto Jr.

SAMUEL A. DINITTO, JR., Chief
Software Technology Division
Command, Control & Communications Directorate

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1993	3. REPORT TYPE AND DATES COVERED In-House Jul 92 - Apr 93

4. TITLE AND SUBTITLE
STATIC TASK ALLOCATION FOR PARALLEL PROCESSING SYSTEMS DURING SOFTWARE DEVELOPMENT

5. FUNDING NUMBERS
PE - 62702F
PR - 5581
TA - 20
WU - 77

6. AUTHOR(S) Richard C. Metzger, Loretta S. Auvil; Rome Laboratory (C3CB), Griffiss AFB NY 13441-4505 and John K. Antonio, AFOSR Summer Research Faculty, Purdue Univerity

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
RL-TR-93-158

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Richard C. Metzger/C3CB (315) 330-7650

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In the sequential world, the mapping of processes to a computer system was not a problem because of the classic Von Newman architecture. However, in the parallel world, the mapping of processes to nodes and the balancing of the load on each node becomes an issue that needs to be addressed. This problem enters software development in the implementation phase and remains an issue through the testing and system integration phases. This report describes a new method developed to improve the mapping on a hypercube system. This new hypersphere mapping approach performs an optimal mapping based on the geometry of the physical system and the communication patterns of the processes. The hypersphere mapper can map processes to nodes where the number of processes exceeds the number of nodes. Other mapping approaches rely on heuristics to cluster the communication graphs into subgraphs to accomplish this mapping. The hypersphere mapper can be instrumental in the implementation phase of software development by specifying an optimal mapping that will improve the performance of the system. This method also provides a transparent programming environment, where the programmer does not need to know about the interconnection network of the physical computer system. The hypersphere mapper was tested using simulated directed and undirected random communication graphs. These tests illustrated improvements in average communication distance between messages and maximum

(Continued)

14. SUBJECT TERMS
Software Development, Parallel Processing, Task Allocation, hypercube

15. NUMBER OF PAGES
40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L

Block 13 (Continued)

utilization of physical node links with some tradeoffs in load balancing. The future of the tool and integration into software engineering tools under development is discussed.

CONTENTS

List of Figures

List of Tables

I. Executive Summary

In the sequential world the mapping of processes to a computer system was not a problem, because of the classic Von Newman architecture. However, in the parallel world, the mapping of processes to nodes and the balancing of the load on each node becomes an issue that needs to be addressed. This problem enters software development in the implementation phase and remains an issue throughout the testing and system integration phases. This report describes a new method developed to improve the mapping on a hypercube system. This new hypersphere mapping approach performs an optimal mapping based on the geometry of the physical system and the communication patterns of the processes in the logical software system. The hypersphere mapper can map processes to nodes where the number of processes exceeds the number of nodes. Other mapping approaches rely on heuristics to cluster the communication graphs into subgraphs to accomplish this mapping. The hypersphere mapper can be instrumental in the implementation phase of software development by specifying an optimal mapping that will improve the performance of the system. This method also provides a transparent programming environment, where the programmer does not need to know about the interconnection network of the physical computer system. The hypersphere mapper was tested using simulated directed and undirected random communication graphs. These tests illustrated improvements in average communication distance between messages and maximum utilization of physical node links with some tradeoffs in load balancing. The future of the tool and integration into software engineering tools under development is discussed.

II. Introduction

A. Motivation

The development of software for the sequential world has been studied extensively. This has been demonstrated by the evolution of software lifecycle models that define the phases and steps of software development. This process has not been specified for parallel software development, but is loosely defined to include the phases of sequential development. The steps a parallel programmer must execute for each phase may differ tremendously, because of the physical system. Since sequential software was based on the von Neuman architecture, the hardware platform was stable and not a great concern. However, parallel architectures vary dramatically and continue to change. Parallel architectures have been divided into two major classes, SIMD(Single-Instruction Multiple-Data) or MIMD (Multiple-Instruction Multiple Data) and are further divided into subclasses with different models of computation. Millions of dollars have been poured into research to create faster parallel machines. In the near future some of these machines will be capable of peak teraflops performance (10^{12} floating point operations per second) [Bel92]. With parallel architectures continuing to change, the limiting factor is not the speed. The true limiting factor is developing software that can achieve these speeds and exploit the power, and concurrency of massively parallel computers. In order to develop this software, techniques and tools must be devised to address the problems that arise in parallel software development and to support the construction of a software lifecycle model that is tailored for parallel machines. Fig. 1 illustrates such a development lifecycle and will be referenced in the context of this report. This lifecycle is meant to highlight the software problems encountered during development of parallel software. At this point in time, it is not all inclusive to the number of problems associated with developing parallel software, but gives the reader the idea of the magnitude of the effort involved.

Parallel programmers face all the inherent problems of sequential development, plus others that were not applicable to the uniprocessor machine. As mentioned in the Executive Summary, the mapping or allocation of processes to nodes and the load balancing on each node is just one of the problems parallel programmers need to address. Some additional problems include, but are not limited to, decomposition of the problem into coherent processes, communication between processes, and synchronization of these processes. Some of these problems depend on the physical architecture and must be a concern for parallel programmers, whereas it was not a concern for sequential programmers. With

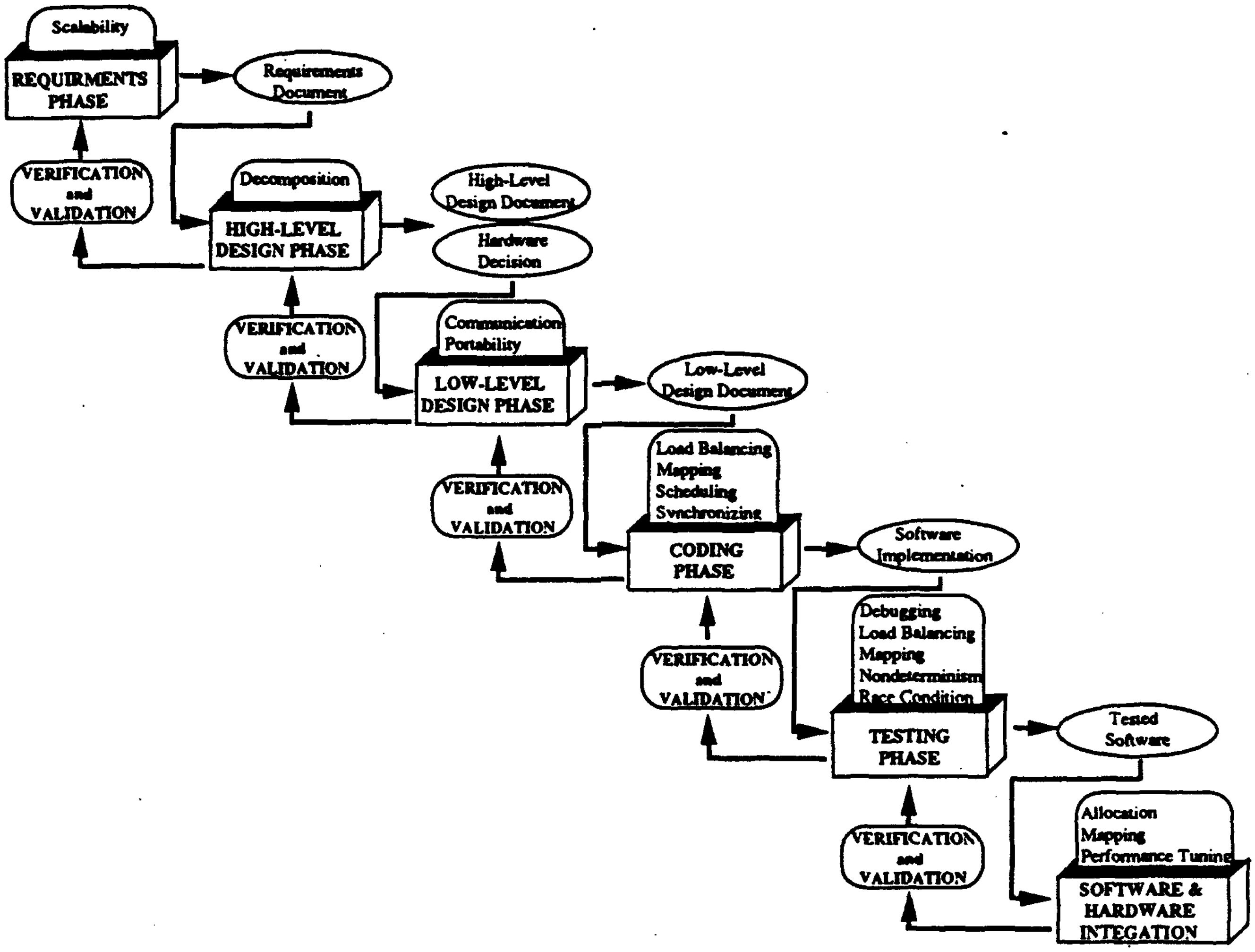

Scalability
REQUIRMENTS PHASE
Requirements Document
VERIFICATION and VALIDATION
Decomposition
HIGH-LEVEL DESIGN PHASE
High-Level Design Document
Hardware Decision
VERIFICATION and VALIDATION
Communication Portability
LOW-LEVEL DESIGN PHASE
Low-Level Design Document
VERIFICATION and VALIDATION
Load Balancing Mapping Scheduling Synchronizing
CODING PHASE
Software Implementation
VERIFICATION and VALIDATION
Debugging Load Balancing Mapping Nondeterminism Race Condition
TESTING PHASE
Tested Software
VERIFICATION and VALIDATION
Allocation Mapping Performance Tuning
SOFTWARE & HARDWARE INTEGATION

Fig. 1. Software development process for parallel processing.

this in mind, it would appear that it is necessary to know the physical architecture at the beginning of software development. This makes the development of transparent parallel software that can achieve a high degree portability across multiple parallel platforms more difficult. The development of techniques and tools that can make parallel software transparent and portable is therfore an important area of research.

A new technique for addressing the mapping and allocation problem in distributed memory hypercube topology systems (commonly called the hypercube embedding problem) while also considering load balancing is discussed in this report. This hypersphere mapping method can be important in the implementation phase of software development by specifying an optimal mapping that will improve the performance of the system. This report discusses the implementation of software on hypercube systems, which have been a popular choice for interconnecting large numbers of processing elements in parallel processing systems. The hypersphere mapper also provides a transparent programming environment, because the programmer does not have to know the underlying interconnection network of the physical architecture.

Effectively mapping a collection of processes onto the nodes of a massively parallel computer is especially important when considering problem domains where the interprocess communication pattern is inherently irregular and/or data dependent. For example, in the area of computational fluid dynamics, some applications require that the solution of a partial differential equation be approximated over an *irregular* grid of discrete points [Fox88]. In other applications, where the processes are *functionally independent*, the associated interprocess communication pattern may also be irregular and possibly data dependent. For example, consider a large embedded information processing system (e.g., a data fusion system) in which the input data comes from a collection of distributed sensors. The processing of the sensor data could be done using functional parallelism, i.e., one process (or perhaps a collection of subprocesses) may be performing FFTs, while other processes are involved in solving systems of linear equations, sorting integers, etc. In such a system, it is possible (perhaps likely) that the communication patterns between the functionally independent processes will be irregular and depend on the values of the input data supplied by the sensors. Therefore, a mapping technique for allocating tasks to processors can be very beneficial, especially if it can do the work for the parallel programmer.

The hypercube embedding problem involves mapping a collection of software tasks, called processes, onto the nodes of a hypercube multiprocessor, so that the available communication resources are effectively utilized. The objective is to determine a mapping that minimizes the average Hamming distance between those pairs of processes that require interprocess communication. This objective is certainly desirable for the case where routing is handled in a store-and-forward message passing fashion because the total delay in sending a message from source to destination is proportional to the number of links traversed. For the case of circuit-switched and virtual cut-through routing schemes, minimizing the average distance between communicating process pairs can reduce the total number of communication links needed to establish the required connections and therefore potentially reduce the latency caused by contention for a limited number of communication resources.

The hypercube embedding problem is known to be NP-hard and several combinatoric heuristics have been proposed and evaluated in the literature. In this report, a nonlinear programming approach is introduced for solving the hypercube embedding problem. The basic idea of the proposed approach is to approximate the discrete space of a n- dimensional hypercube, i.e., $\{z : z \in \{0,1\}^n\}$, with the continuous space of a n-dimensional hypersphere, i.e., $\{x : x \in \Re^n \;\&\; ||x||^2 = 1\}$. The mapping problem is initially solved in the continuous domain by employing the gradient projection technique to a continuously differentiable objective function, where the primary objective is to minimize the average squared Euclidean distance between communicating processes. The optimal process "locations" from the solution of the continuous hypersphere mapping problem are then discretized onto the n-dimensional hypercube. The proposed approach can solve, directly, the problem of mapping P processes onto N nodes for the general case where $P > N$. In contrast, competing embedding heuristics from the literature can produce only one-to-one mappings and therefore cannot be directly applied when $P > N$.

B. Related Work

The hypercube embedding problem has been studied extensively in the past and a variety of mapping heuristics and theoretical results have been reported in the literature. [CSG89] evaluates twelve hypercube mapping heuristics in terms of mapping quality and run time of the mapping algorithms themselves for several classes of interprocess communication patterns. In additional papers, the fundamental theoretical results have been established for the case where the communication patterns are assumed to have regular structures such as grids [BMS88], trees [Wu85], binary trees [Wag89], and hypercubes [Bha80].

One potential drawback of previous embedding heuristics is that they produce one-to-one mappings. In practical applications, the requirement may arise to map P processes onto N nodes, where $P > N$; thus, many-to-one mappings are generally required. It appears that all the known mapping heuristics (including the twelve evaluated in [CSG89]) assume $P \leq N$. A straightforward way to overcome the one-to-one limitation of these mapping heuristics is to initially cluster the P processes into N (or fewer) clusters and then execute the mapping heuristic based on the intercluster communication pattern. Since the clustering problem is generally NP-complete, an additional heuristic would be required to do the initial clustering before actually executing the mapping heuristic. One of the advantages of the hypersphere mapping algorithm is that the case of $P > N$ is handled directly, because the produced mapping solution is not limited to the class of one-to-one functions.

C. Organization of the Paper

In Section II, the general hypercube embedding problem is formulated in mathematical terms. A brief overview of the proposed algorithm, called hypersphere mapper, is given in Section III. The hypersphere mapper utilizes an iterative gradient descent technique known as the gradient projection method, so a general overview of the gradient projection method is provided in Section IV. Section V contains the detailed description of hypersphere mapper. In Section VI, extensive simulation studies for hypersphere mapper are conducted for the case of randomly generated process communication patterns. Section VII presents the conclusions and future work.

III. The Hypercube Embedding Problem

Mapping a set of P processes, denoted $\mathcal{P} = \{0, 1, \ldots, P-1\}$, onto an $N = 2^n$ node (i.e., n-dimensional) hypercube is viewed as a problem of determining a collection of P binary n-vectors, $z_i \in \{z : z \in \{0,1\}^n\}$, $i \in \mathcal{P}$, where the components of z_i comprise the binary address of the node that process i is to be mapped to. For instance, for the case $n = 4$, $z_0 = [\ 1 \ \ 1 \ \ 0 \ \ 0\ ]^T$ indicates that process 0 is to be mapped onto node 3.

Let $d_H(z_i, z_j)$ denote the Hamming distance between nodes with addresses z_i and z_j, where the Hamming distance equals the number of differing binary components associated with the vectors z_i and z_j. For example, $d_H([\ 1 \ \ 0 \ \ 1 \ \ 0\ ]^T, [\ 1 \ \ 1 \ \ 0 \ \ 0\ ]^T) = 2$. Let $\mathcal{W}$ denote the set of pairs of processes that require communication. Thus, $\mathcal{W} = \{(i,j) : i, j \in \mathcal{P}$ & process i communicates with process $j\}$.

Therefore, for a given set $\mathcal{W}$ (i.e., a given communication pattern), the hypercube embedding problem involves finding P binary n-vectors, denoted by $z_0, z_1, \ldots, z_{P-1}$, that minimize

$$f_H(z) = \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} d_H(z_i, z_j) . \qquad (II.1)$$

The standard hypercube embedding problem assumes that $P \leq N$ and that the mapping is a one-to-one-function, i.e., $z_i \neq z_j$, for all $i, j \in \mathcal{P}$, with $i \neq j$. [CKV87] has been proven that the standard embedding problem is NP-hard.

IV. Overview of Hypersphere Mapper

The main premise of the hypersphere mapper approach is to approximate the discrete space of the hypercube, i.e., $\{z : z \in \{0,1\}^n\}$, with a continuous n-dimensional (unit radius) hypersphere, i.e., $\{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$. The mapping problem in the continuous domain is to determine a collection of P real n-vectors, $x_i \in \{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$, $i \in \mathcal{P}$. The advantage of the continuous domain is that techniques from nonlinear programming can be employed to solve a constrained optimization problem. In particular, by defining a continuously differentiable objective function, the values of x_i are updated in an iterative fashion by using the gradient projection technique. Geometrically, the vectors x_i are points that reside on the surface of a hypersphere in the continuous domain of $\Re^n$. The application of the iterative gradient projection technique acts to migrate these points around the surface of the hypersphere so as to minimize a desired objective function.

The objective function used by hypersphere mapper comprises two components: the first component is the average squared Euclidean distance between communicating pairs of processes, the second component is the average of the inverted squared Euclidean distance between every pair of processes. The first component acts to bring pairs of communicating processes close together; the second component acts to "spread-out" the process locations (and prevent any pair of processes from residing at the same point on the continuous hypersphere).

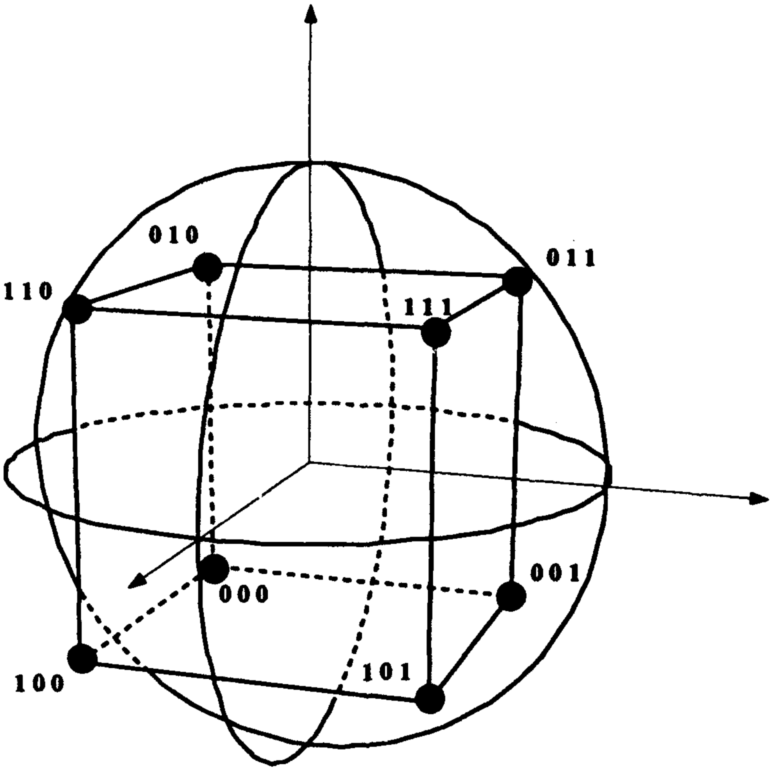


Fig. 2. A geometric view of how points from the surface of a continuous hypersphere are discretized onto the nodes of a discrete hypercube.

Once a solution is obtained on the continuous hypersphere, the solution vectors, denoted by x_i^*, are discretized onto the n-dimensional binary hypercube according to the sign of each component. For example, in three dimensions, the continuous vector $x_0^* = [\ 0.4000 \quad -0.5000 \quad 0.7681\ ]^T$ discretizes to $z_0^* = [\ 1 \quad 0 \quad 1\ ]^T$, which indicates that process 0 is to be mapped onto node 5. Fig. 2 depicts a geometric interpretation of how points on the surface of the continuous hypersphere are discretized onto the underlying hypercube. All continuous points belonging to the same sector are discretized onto the corresponding node. For the three dimensional case shown, note that there are eight sectors and eight nodes. In the general n-dimensional case, there are 2^n sectors and and 2^n nodes.

V. THE GRADIENT PROJECTION METHOD

The gradient projection method is an iterative gradient descent technique for solving constrained optimization problems. Each iteration of the gradient projection method comprises two parts: (1) updating the values of the variables by moving in the direction of the negative gradient of the objective function and (2) orthogonally projecting the values of the variables onto the constraint space.

Consider the following general constrained optimization problem:

$$\text{minimize } \; f(x) \tag{IV.1}$$

$$\text{subject to } \; g(x) = 0. \tag{IV.2}$$

The first part of the gradient projection method is to update x by moving in the direction of the negative gradient of f. Thus, letting $x^{(k)}$ denote the value of the vector x at iteration k, the first part of the gradient projection method is given by

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x)|_{x=x^{(k)}}, \tag{IV.3}$$

where α is a positive scalar called the stepsize.

The second part of the gradient projection method is to orthogonally project $x^{(k+1)}$ onto the constraint space $g(x) = 0$. This projection operation is denoted by

$$x^{(k+1)} \leftarrow [x^{(k+1)}]^{g(x)=0}. \tag{IV.4}$$

The mathematical description of precisely how to do the projection for general constraint surfaces shall not be included here. The interested reader is referred to [BaS79]. A geometric view of an iteration from the gradient projection method being applied to a general constrained optimization problem is depicted in Fig. 3. The dashed lines represent constant value contours for the objective function $f(x)$, i.e., $f(x) = C_1$ and $f(x) = C_2$. The value of $x^{(k)}$ is initially updated by moving in the direction of the negative gradient,

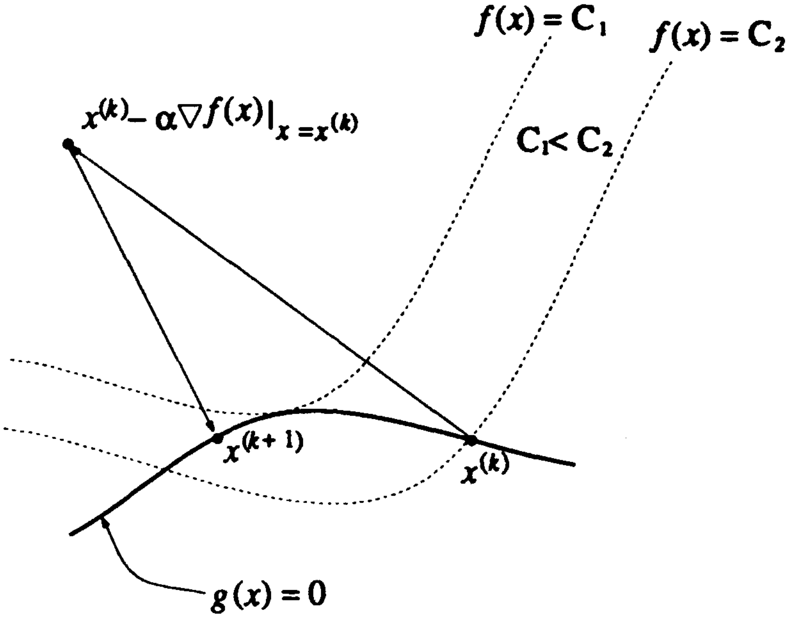


Fig. 3. A geometric view of an iteration of the gradient projection method.

i.e., $x^{(k)}$ is updated to the point $x^{(k)} - \alpha\nabla f(x)|_{x=x^{(k)}}$. Then, the updated value is projected back to the constraint surface $g(x) = 0$.

VI. Detailed Description of Hypersphere Mapper

The detailed description of hypersphere mapper is divided into three main parts. First, a constrained optimization problem is formulated for the problem of mapping processes onto the surface of a continuous n-dimensional hypersphere. Second, the gradient projection technique is applied as a means of solving the proposed constrained optimization problem. Finally, the method of discretizing the continuous process "locations" (from the solution of the constrained optimization problem) is described. After presenting the detailed description of hypersphere mapper, an illustrative example application is given. The final subsection describes a simple technique for incrementally spreading-out the hypersphere mapper solution so as to obtain more uniform mappings.

A. Formulating the Constrained Optimization Problem

In the domain of the continuous hypersphere, the underlying objective is to minimize the average squared Euclidean distance between pairs of communicating processes, where the processes are located on the surface of an n-dimensional hypersphere in $\Re^n$. Thus, given $\mathcal{W}$ (i.e., the set of pairs of processes that require communication), the problem is to find P real n-vectors, $x_i \in \{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$, $i \in \mathcal{P}$, that minimize

$$\frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2, \qquad (V.1)$$

where $x_i = (\ x_{i,0}\ \ x_{i,1}\ \ \cdots\ \ x_{i,n-1})^T$ and $||x_i - x_j||^2 = \sum_{\ell=0}^{n-1}(x_{i,\ell} - x_{j,\ell})^2$.

It is also desirable to enforce that $x_i \neq x_j$, for all $i, j \in \mathcal{P}$, with $i \neq j$. To enforce this constraint, the following penalty term is proposed:

$$\frac{2}{P(P-1)} \sum_{i=0}^{P-2} \sum_{j=i+1}^{P-1} \frac{1}{||x_i - x_j||^2}, \qquad (V.2)$$

which is the average of the inverted squared Euclidean distance between all pairs of processes. The desired objective function comes by adding Equations $V.1$ and $V.2$:

$$\frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{P(P-1)} \sum_{i=0}^{P-2} \sum_{j=i+1}^{P-1} \frac{1}{||x_i - x_j||^2}. \qquad (V.3)$$

The first term in Equation $V.3$ acts to bring the communicating pairs of processes close together while the second terms ensures that no two processes reside at the same position.

Combining the objective function of Equation $V.3$ with the constraint that the processes must be located on the surface of an n-dimensional hypersphere results in the following constrained optimization problem:

$$\text{minimize} \ \left\{ \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{P(P-1)} \sum_{i=0}^{P-2} \sum_{j=i+1}^{P-1} \frac{1}{||x_i - x_j||^2} \right\} \qquad (V.4)$$

$$\text{subject to} \ \ ||x_i||^2 = 1, \ \ \text{for all } i \in \mathcal{P}. \qquad (V.5)$$

B. *Application of the Gradient Projection Method*

Note that the constrained optimization problem of Equations $V.4$ and $V.5$ can be expressed as:

$$\text{minimize} \;\; f(x) \tag{V.6}$$

$$\text{subject to} \;\; g(x) = 0, \tag{V.7}$$

where

$$f(x) = \left\{ \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{P(P-1)} \sum_{i=0}^{P-2} \sum_{j=i+1}^{P-1} \frac{1}{||x_i - x_j||^2} \right\}, \tag{V.8}$$

$$g(x) = \begin{pmatrix} ||x_0||^2 - 1 \\ ||x_1||^2 - 1 \\ \vdots \\ ||x_{P-1}||^2 - 1 \end{pmatrix}, \tag{V.9}$$

and

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{P-1} \end{pmatrix}. \tag{V.10}$$

Derivation of the Gradient

Recall from Equation $IV.3$ that the first part of each iteration of the gradient projection method requires the gradient of $f(x)$, i.e., $\nabla f(x)$. By differentiating terms from $f(x)$, it can be shown that

$$\frac{\partial}{\partial x_i}\left\{||x_i - x_j||^2\right\} = 2(x_i - x_j),$$

$$\frac{\partial}{\partial x_j}\left\{||x_i - x_j||^2\right\} = 2(x_j - x_i),$$

$$\frac{\partial}{\partial x_i}\left\{\frac{1}{||x_i - x_j||^2}\right\} = \frac{2}{||x_i - x_j||^4}(x_j - x_i),$$

and

$$\frac{\partial}{\partial x_j}\left\{\frac{1}{||x_i - x_j||^2}\right\} = \frac{2}{||x_i - x_j||^4}(x_i - x_j).$$

Define $\mathcal{W}(i) = \{j : j \in \mathcal{P}$ & process i and process j communicate $\}$. Also, define $\mathcal{P}(i) = \mathcal{P} - \{i\}$. With these definitions, the gradient of $f(x)$ is expressed as

$$\nabla f(x) = \begin{pmatrix} \frac{2}{|W|}\sum_{j\in\mathcal{W}(0)}(x_0 - x_j) + \frac{4}{P(P-1)}\sum_{j\in\mathcal{P}(0)}\frac{(x_j - x_0)}{\|x_0 - x_j\|^4} \\ \frac{2}{|W|}\sum_{j\in\mathcal{W}(1)}(x_1 - x_j) + \frac{4}{P(P-1)}\sum_{j\in\mathcal{P}(1)}\frac{(x_j - x_1)}{\|x_1 - x_j\|^4} \\ \vdots \\ \frac{2}{|W|}\sum_{j\in\mathcal{W}(P-1)}(x_{P-1} - x_j) + \frac{4}{P(P-1)}\sum_{j\in\mathcal{P}(P-1)}\frac{(x_j - x_{P-1})}{\|x_{P-1} - x_j\|^4} \end{pmatrix}. \qquad (V.11)$$

Thus, the first part of the iteration for the gradient projection method (i.e., moving in the direction of the negative gradient) is given by

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha\nabla f(x)|_{x=x^{(k)}}, \qquad (V.12)$$

where $\nabla f(x)|_{x=x^{(k)}}$ is computed according to Equation $V.11$.

Derivation of tne Projection

After the update of Equation $V.12$ is computed, the second part of the gradient projection method requires that the resultant $x^{(k+1)}$ be projected back to the surface $g(x) = 0$. This amounts to simply normalizing each x_i to be unit length (in the Euclidean sense). Thus, the projection operator is defined by

$$[x]^{g(x)=0} = \begin{pmatrix} \frac{x_0}{\|x_0\|} \\ \frac{x_1}{\|x_1\|} \\ \vdots \\ \frac{x_{P-1}}{\|x_{P-1}\|} \end{pmatrix}. \qquad (V.13)$$

So, the second part of the iteration for the gradient projection method is given by

$$x^{(k+1)} \leftarrow [x^{(k+1)}]^{g(x)=0}, \qquad (V.14)$$

where $[x^{(k+1)}]^{g(x)=0}$ is computed according to Equation $V.13$.

After a sufficient number of iterations, the gradient projection method converges to a fixed-point, i.e., the vector $x^{(k)}$ converges to a fixed point x^* as k increases.

C. Discretizing onto the Hypercube

The final part of hypersphere mapper is to convert the solution vector x^*, which has continuous real components, to z^*, which has discrete components in the set $\{0,1\}$. The conversion is done by simply discretizing each component of x (to either zero or one) according to their signs. Formally, for any $t \in \Re$, the unit step function is defined as

$$u(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases}$$

Therefore, the discretization is defined by

$$z^* = u(x^*),$$

where it is understood that $u(\cdot)$ is applied to each component of x^*.

D. An Illustrative Example

The goal here is to illustrate how the process locations move around the surface of the hypersphere from one iteration to the next. Of particular interest is to show how the relative locations of the processes (for a given communication pattern, $\mathcal{W}$) affect the iterative migration to the fixed point.

Consider the problem of mapping eight processes, $\mathcal{P} = \{0, 1, \ldots, 7\}$, onto the nodes of a 3-dimensional hypercube. Assume the communication pattern is $\mathcal{W} = \{(0,4), (0,7), (1,7), (1,6), (2,4), (2,5), (3,5), (3,6)\}$. A graphical view of the assumed process communication pattern, i.e., $\mathcal{W}$, is shown in Fig. 4. The resulting iteration data from hypersphere mapper is given in Table I. Fig. 5 shows a 3-dimensional perspective of the position vectors for iterations $k = 0, 1, \ldots, 5$. The positions are depicted by the dark vectors and the underlying hypersphere/hypercube structure is superimposed in gray.

From Table I, note that the initial values of the continuous location vectors (i.e., the values of $x_i^{(0)}$, $i = 0, 1, \ldots, 7$) were selected so that process i is (initially) mapped to node

Table I. Iterations from Hypersphere Mapper

k	x_0^k	x_1^k	x_2^k	x_3^k	x_4^k	x_5^k	x_6^k	x_7^k	$f(x^k)$
	z_0^k	z_1^k	z_2^k	z_3^k	z_4^k	z_5^k	z_6^k	z_7^k	$f_H(z^k)$
0	−0.66 −0.56 −0.50	0.61 −0.75 −0.26	−0.34 0.07 −0.94	0.66 0.55 −0.50	−0.03 −0.80 0.59	0.09 −0.91 0.41	−0.45 0.61 0.65	0.61 0.55 0.57	4.12
	0 0 0	1 0 0	0 1 0	1 1 0	0 0 1	1 0 1	0 1 1	1 1 1	2.25
1	−0.66 −0.68 −0.32	0.78 −0.61 −0.14	−0.36 −0.16 −0.92	0.69 0.59 −0.40	−0.51 0.15 0.84	0.46 −0.62 −0.63	−0.33 0.70 0.63	0.73 0.41 0.54	3.06
	0 0 0	1 0 0	0 0 0	1 1 0	0 1 1	1 0 0	0 1 1	1 1 1	2.00
2	−0.73 −0.68 −0.05	0.88 −0.42 0.22	−0.38 −0.17 −0.91	0.68 0.61 −0.42	−0.71 −0.15 0.69	0.35 −0.47 −0.81	−0.07 0.87 0.48	0.81 0.21 0.54	2.31
	0 0 0	1 0 1	0 0 0	1 1 0	0 0 1	1 0 0	0 1 1	1 1 1	1.50
3	−0.72 −0.69 0.13	0.89 −0.32 0.31	−0.46 −0.19 −0.87	0.64 0.63 −0.45	−0.82 −0.24 0.53	0.38 −0.36 −0.85	0.14 0.89 0.44	0.78 0.11 0.61	2.06
	0 0 1	1 0 1	0 0 0	1 1 0	0 0 1	1 0 0	1 1 1	1 1 1	1.00
4	−0.62 −0.76 0.18	0.90 −0.34 0.28	−0.52 −0.23 −0.82	0.62 0.63 −0.47	−0.87 −0.19 0.45	0.37 −0.27 −0.89	0.30 0.87 0.39	0.66 0.13 0.74	1.89
	0 0 1	1 0 1	0 0 0	1 1 0	0 0 1	1 0 0	1 1 1	1 1 1	1.00
5	−0.53 −0.79 0.30	0.92 −0.24 0.33	−0.56 −0.25 −0.77	0.61 0.63 −0.49	−0.92 −0.19 0.35	0.35 −0.19 −0.91	0.42 0.85 0.32	0.59 −0.01 0.81	1.70
	0 0 1	1 0 1	0 0 0	1 1 0	0 0 1	1 0 0	1 1 1	1 0 1	0.75
9	−0.39 −0.70 0.60	0.91 −0.01 0.42	−0.62 −0.27 −0.74	0.57 0.59 −0.58	−0.94 −0.33 0.07	0.19 −0.05 −0.98	0.66 0.72 0.20	0.33 −0.34 0.88	1.38
	0 0 1	1 0 1	0 0 0	1 1 0	0 0 1	1 0 0	1 1 1	1 0 1	0.75
10	−0.39 −0.68 0.61	0.89 0.01 0.45	−0.62 −0.27 −0.74	0.56 0.58 −0.59	−0.93 −0.33 0.05	0.16 −0.03 −0.99	0.68 0.71 0.18	0.32 −0.36 0.88	1.37
	0 0 1	1 1 1	0 0 0	1 1 0	0 0 1	1 0 0	1 1 1	1 0 1	0.75

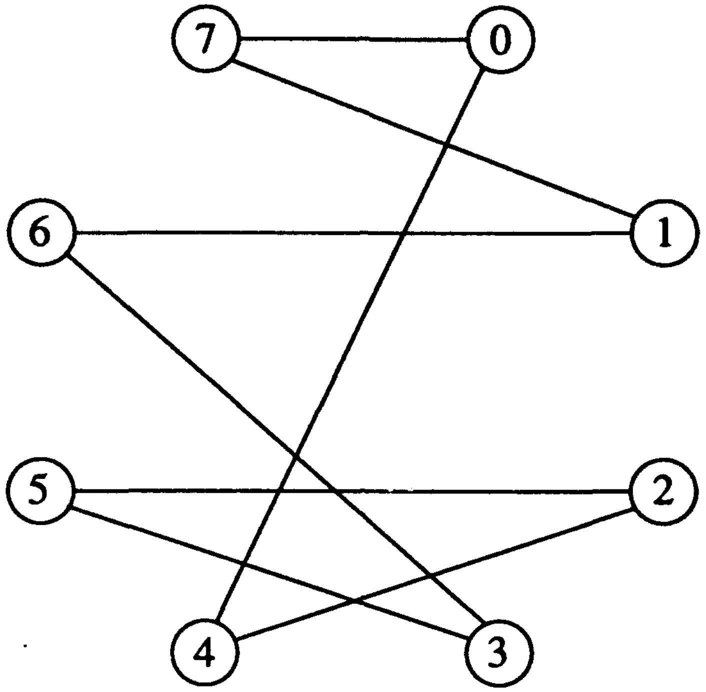
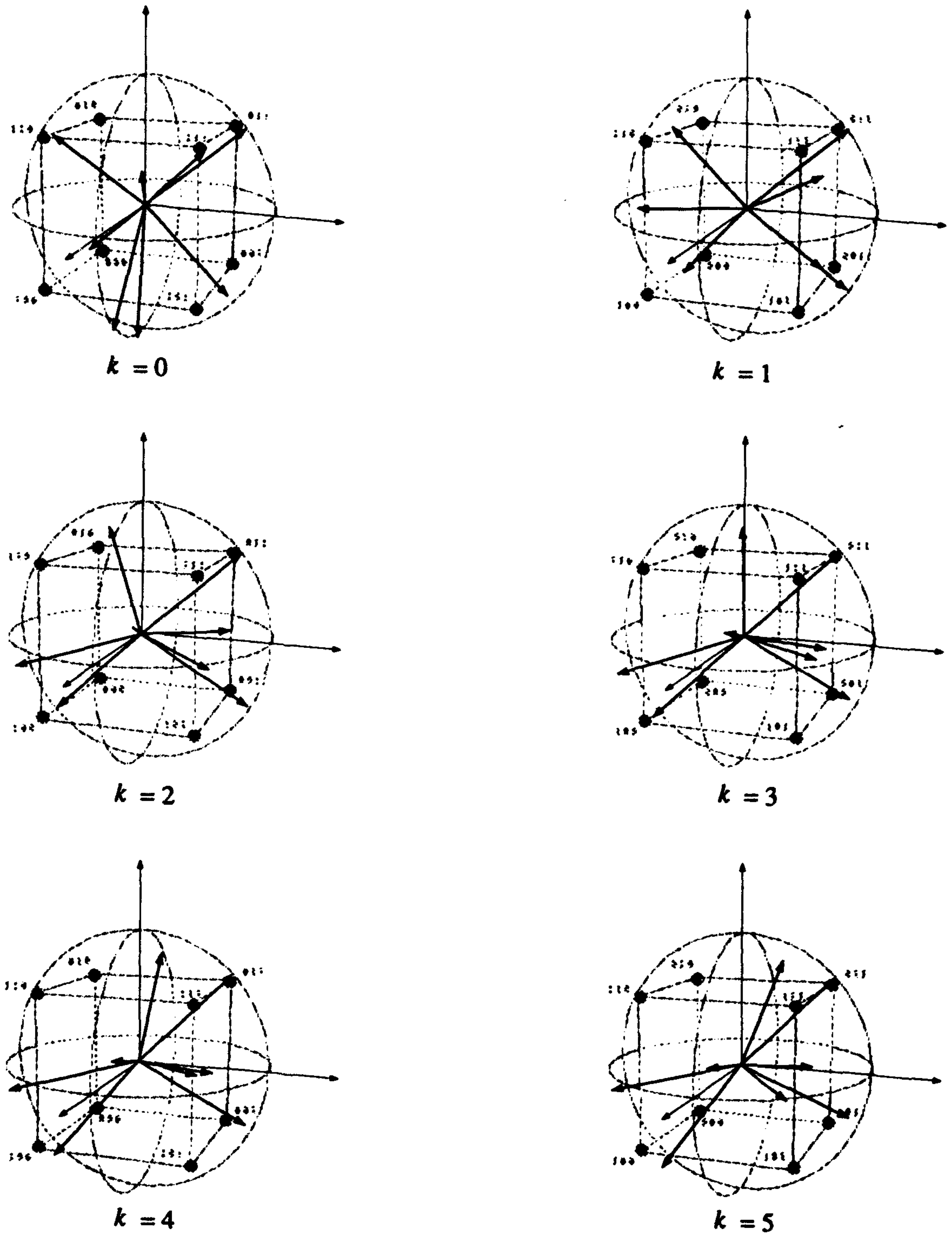


Fig. 4. Graphical view of the example process communication pattern $\mathcal{W} = \{(0,4),(0,7),(1,7),(1,6),(2,4),(2,5),(3,5),(3,6)\}$.

i, $i = 0, 1, \ldots, 7$. After each iteration k, the resulting values of the continuous location vectors, $x_i^{(k)}$, and the associated discretized location vectors, $z_i^{(k)}$, are given. Also, the corresponding values of the continuous objective function, $f(x^{(k)})$ (refer to Equation V.8), and the associated average Hamming distance, $f_H(z^{(k)})$ (refer to Equation II.1), are given. The fact that the value of $f_H(x^{(k)})$ decreases as k increases indicates that the continuous objective function does indeed capture the essence of the discrete mapping problem.

Note that the process to node mapping produced by hypersphere mapper (taken from iteration $k = 10$ of Table I) is:

process 0 → node 4
process 1 → node 7
process 2 → node 0
process 3 → node 3
process 4 → node 4
process 5 → node 1
process 6 → node 7
process 7 → node 5,



$k=0$

$k=1$

$k=2$

$k=3$

$k=4$

$k=5$

Fig. 5. A geometric view of how the position vectors migrate around the hypersphere during the first five iterations.

which is *not* a one-to-one mapping because two processes are mapped to node 4 and two processes are mapped to node 7 (and no processes are mapped to node 2 nor node 6). The average Hamming distance between communicating processes for this mapping is 0.75.

It can be verified (through an exhaustive search, for example) that an optimal *one-to-one* mapping is given by:

process 0 → node 5
process 1 → node 6
process 2 → node 0
process 3 → node 3
process 4 → node 4
process 5 → node 1
process 6 → node 2
process 7 → node 7.

The above optimal one-to-one mapping has an associated average Hamming distance of 1.00. Thus, for this example, the solution from hypersphere mapper produces a mapping that is superior, in terms of the average Hamming distance, to the best possible one-to-one mapping (i.e., 0.75 versus 1.00) at the "expense" of not being one-to-one.

E. Converting Hypersphere Mapper Solutions into One-To-One Mappings

An intuitive approach is introduced here for modifying the continuous solution produced by hypersphere mapper so that after discretization, the resulting mapping will be one-to-one (or to within a specified degree of closeness to being one-to-one). The basic idea of the approach is the incrementally spread continuous process locations out of "over-populated" sectors and into nearby "under-populated" sectors. The detailed description of the approach requires the following definition.

In the space of a continuous n-dimensional hypersphere, let $\underline{c_i} = (c_{i,0} \ c_{i,1} \ \ldots \ c_{i,n-1})^T \in \{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$ denote the center of sector i, which is defined for all i =

$0, 1, \ldots, 2^n - 1$ as follows:

$$c_{i,j} = \begin{cases} \frac{1}{\sqrt{n}}, & \text{if the } j\text{th bit of the binary representation of } i \text{ is unity} \\ -\frac{1}{\sqrt{n}}, & \text{if the } j\text{th bit of the binary representation of } i \text{ is zero.} \end{cases}$$

For instance, for the 3-dimensional case, $c_0 = (-\frac{1}{\sqrt{3}} \quad -\frac{1}{\sqrt{3}} \quad -\frac{1}{\sqrt{3}})^T$ and $c_6 = (-\frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}})^T$. The center vectors define the geometric centers for each of the 2^n sectors associated with an n-dimensional hypersphere.

The algorithm for incrementally spreading out the continuous solution from hypersphere mapper operates as follows. First, those sectors having strictly more than $\lceil \frac{P}{N} \rceil$ processes residing in them are flagged as being "over-populated" and those having no more than $\lfloor \frac{P}{N} \rfloor$ processes residing in them are flagged as being "under-populated." The spreading procedure takes place in n consecutive phases, denoted by phase 1 through phase n. During phase i, those position vectors residing in over-populated sectors that are within a Euclidean distance of $2\sqrt{\frac{i}{n}}$ from the center of an under-populated sector are moved to that under-populated sector. Immediately after a position vector is moved from an over-populated sector to an under-populated sector, the status of the associated sectors' population is updated. Because the diameter of the continuous hypersphere is 2 (in the Euclidean sense), after phase n of the procedure, it is clear that there will be no over-populated nor under-populated sectors. Also, the mapping associated with phase j of the spreading procedure is generically a more uniform mapping than the one associated with phase i, for all $i < j$.

The spreading procedure was applied to the illustrative example of the previous subsection (i.e., the problem of mapping 8 processes onto 8 nodes assuming the communication pattern $\mathcal{W}$ of Fig. 4). The results of applying the spreading procedure are depicted in Fig. 6. Without application of the spreading procedure, note that processes 0 and 4 are both mapped to node 4, processes 1 and 6 are both mapped to node 7, and no processes are mapped to either node 2 nor 6. Phase 1 of spreading moves process 4 from the over-populated node 4 onto the under-populated node 6. Phase 2 of spreading moves process 6 from the over-populated node 7 onto the under-populated node 2.

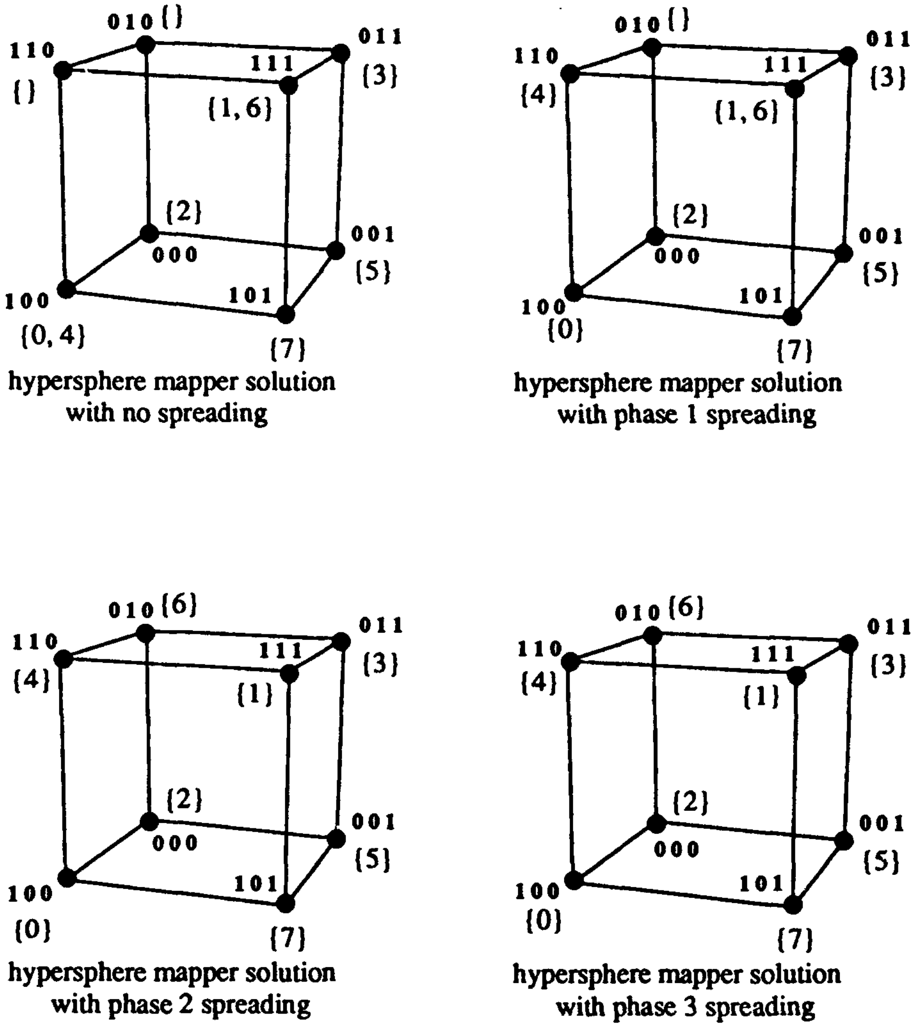
VII. Evaluation of Hypersphere Mapper

A. Comparison with Other Hypercube Embedding Heuristics

In reference [GSG89], evaluations and comparisons (based on simulation studies) are reported for twelve hypercube embedding heuristics. Several classes of process communication patterns are assumed, including random patterns, permuted hypercube patterns, and tree patterns. Of the heuristics evaluated (and for all of the assumed process communication patterns), the simulated annealing heuristic developed in [CSG89] produced mappings with the smallest average Hamming distance.[1] In contrast, a greedy algorithm of [ChG88] generally produced the poorest mappings (excluding the so-called default or random mapping schemes, which map processes to nodes independent of the given process communication pattern). It is also reported that the (typical) running time for the simulated annealing algorithm is around four orders of magnitude larger than that of the greedy algorithm. Both the quality of the mappings and the running times associated with the other ten heuristics were shown to lie within the bounds defined by the simulated annealing and greedy algorithms. For a detailed description, evaluation, and comparison of the twelve heuristics, refer to [CSG89].

In the present report, simulation studies are used to provide an indication of how the quality of mappings produced by hypersphere mapper compare to the quality of mappings produced by the other known embedding heuristics. The simulation study carried out in this subsection involves mapping 128 processes onto the 128 nodes of a 7-dimensional hypercube. The assumed communication pattern is generated by selecting process source-destination pairs at random. The expected number of randomly selected

[1] Refer to [KGV83] for general background material on the simulated annealing approach.



hypersphere mapper solution
with no spreading

hypersphere mapper solution
with phase 1 spreading

hypersphere mapper solution
with phase 2 spreading

hypersphere mapper solution
with phase 3 spreading

Fig. 6. Applying the spreading procedure to convert the solution from hypersphere mapper into a one-to-one function. The notation "$\{A, B\}$" beside node address "*abc*" means that processes A and B are mapped onto the node labeled *abc*.

source-destination pairs is set at 448, i.e., $E[|\mathcal{W}|] = 448$. This exact same simulation study was one of the experiments used in evaluating the heuristics in [CSG89].

The results of simulation studies for the hypersphere mapper, simulated annealing, greedy, and default algorithms (averaged over 100 runs) are summarized in Table II. The column labeled $\hat{f}_H(z)$ denotes the average Hamming distance between mapped communicating processes. The column labeled $\hat{\sigma}^2_{\text{pmn}}$ denotes the sample variance of the number of processes mapped to each node, which is defined by

$$\hat{\sigma}^2_{\text{pmn}} = \frac{1}{N} \sum_{i=0}^{N} \left(\text{pmn}(i) - \frac{P}{N} \right)^2 ,$$

where pmn(i) denotes the number of processes mapped to node i. The term $\frac{P}{N}$ represents the average number of nodes mapped to each node, which equals unity for the simulation of this subsection because there are $P = 128$ processes being mapped onto $N = 128$ nodes. (In general, the average number of processes mapped onto each node equals $\frac{P}{N}$, regardless of the mapping).

From Table II, note that the value of $\hat{\sigma}^2_{\text{pmn}}$ is nonzero only for the hypersphere mapper because the mappings produced by hypersphere mapper are not generically one-to-one. On the other hand, the competing algorithms always produce one-to-one mappings and thus have zero values for $\hat{\sigma}^2_{\text{pmn}}$ (i.e., exactly one process is mapped to each node, thus the variance of the number of processes mapped to each node is zero). The results tabulated for the simulated annealing and greedy algorithms were taken from reference [CSG89].[2] The last row of Table II gives the results of the default mapping algorithm, which simply maps process i to node i, for all $i = 0, 1, \ldots, 127$.

With regard to average Hamming distance, the mappings produced by hypersphere mapper are indeed superior to those produced by the simulated annealing algorithm (i.e.,

[2] The greedy algorithm was coded and simulated using our randomly generated communication patterns. The value of average Hamming distance produced by our simulation of the greedy algorithm was within 2% of the value reported in [CSG89].

TABLE II. COMAPRISON BETWEEN HYPERSPHERE MAPPER AND OTHER HEURISTICS: 128 PROCESSES ONTO 128 NODES WITH $E[|\mathcal{W}|] = 448$

Algorithm	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\mathrm{pmn}}$
Hypersphere Mapper	1.889	1.68
Simulated Annealing	2.042	0.00
Greedy	2.867	0.00
Default	3.503	0.00

1.889 versus 2.042). However, the mappings produced by hypersphere mapper are not generically one-to-one; thus, the corresponding value of $\hat{\sigma}^2_{\mathrm{pmn}}$ is not equal to zero.

Table III shows the effect of applying the spreading procedure to the hypersphere mapper solution. The average Hamming distance of the mapping produced after spreading phase 1 is slightly better than that of the simulated annealing algorithm (i.e., 2.020 versus 2.042) and the value of the variance of the number of processes mapped to each node, relative to the hypersphere mapper solution with no spreading, is reduced by around 60% (i.e., from 1.68 down to 1.07). It is also noted that the *one-to-one* mapping produced after spreading phase 7 is significantly better than the mapping produced by the greedy algorithm (i.e., 2.587 versus 2.867).

B. The Case of Having More Processes than Nodes

A primary attribute of hypersphere mapper is that problems involving more processes than nodes (i.e., $P > N$) can be solved directly. In contrast, the competing hypercube embedding heuristics (including the twelve evaluated in [CSG89]) require $P \leq N$.

Simulation results are reported here for the case of mapping 256 processes onto 64 nodes. The assumed process communication patterns are randomly generated (with a specified expected number of source-destination pairs). Simulation studies are conducted for cases where the expected number of source-destination pairs is 128, 256, 512, and

TABLE III. APPLICATION OF THE SPREADING PROCEDURE TO HYPERSPHERE MAPPER: 128 PROCESSES ONTO 128 NODES WITH $E[|\mathcal{W}|] = 448$

spreading	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\mathrm{pmn}}$
none	1.889	1.68
phase 1	2.020	1.07
phase 2	2.283	0.40
phase 3	2.440	0.15
phase 4	2.524	0.06
phase 5	2.558	0.03
phase 6	2.580	0.01
phase 7	2.587	0.00

1024. Table IV shows the results of the simulation studies in terms of average Hamming distance, $\hat{f}_h(z)$, and the sample variance of the number of processes mapped onto each node, $\hat{\sigma}^2_{\mathrm{pmn}}$. As would be expected, the average Hamming distance increases as the expected number of source-destination pairs increases. Also, for a fixed value of the expected number of source-destination pairs, the average Hamming distance increases with successive applications of the spreading procedure. Fig. 7 shows histograms for the number of processes mapped to each node for the case of 256 source-destination pairs. The histogram for the solution from hypersphere mapper (without application of the spreading procedure) has the largest sample variance, $\hat{\sigma}^2_{\mathrm{pmn}} = 4.10$. At the other extreme, the histogram associated with the mappings produced after spreading phase 6 has the smallest sample variance, $\hat{\sigma}^2_{\mathrm{pmn}} = 0.00$. The mappings produced by the various spreading phases provide levels of tradeoff between the values of $\hat{f}_H(z)$ and $\hat{\sigma}^2_{\mathrm{pmn}}$. In practical systems, the latency caused by contention for communication resources can be more of a throughput bottleneck than the level of computational load balance. Therefore, mappings with "perfect" load balance (i.e., a minimal value for $\hat{\sigma}^2_{\mathrm{pmn}}$) may not always be preferable.

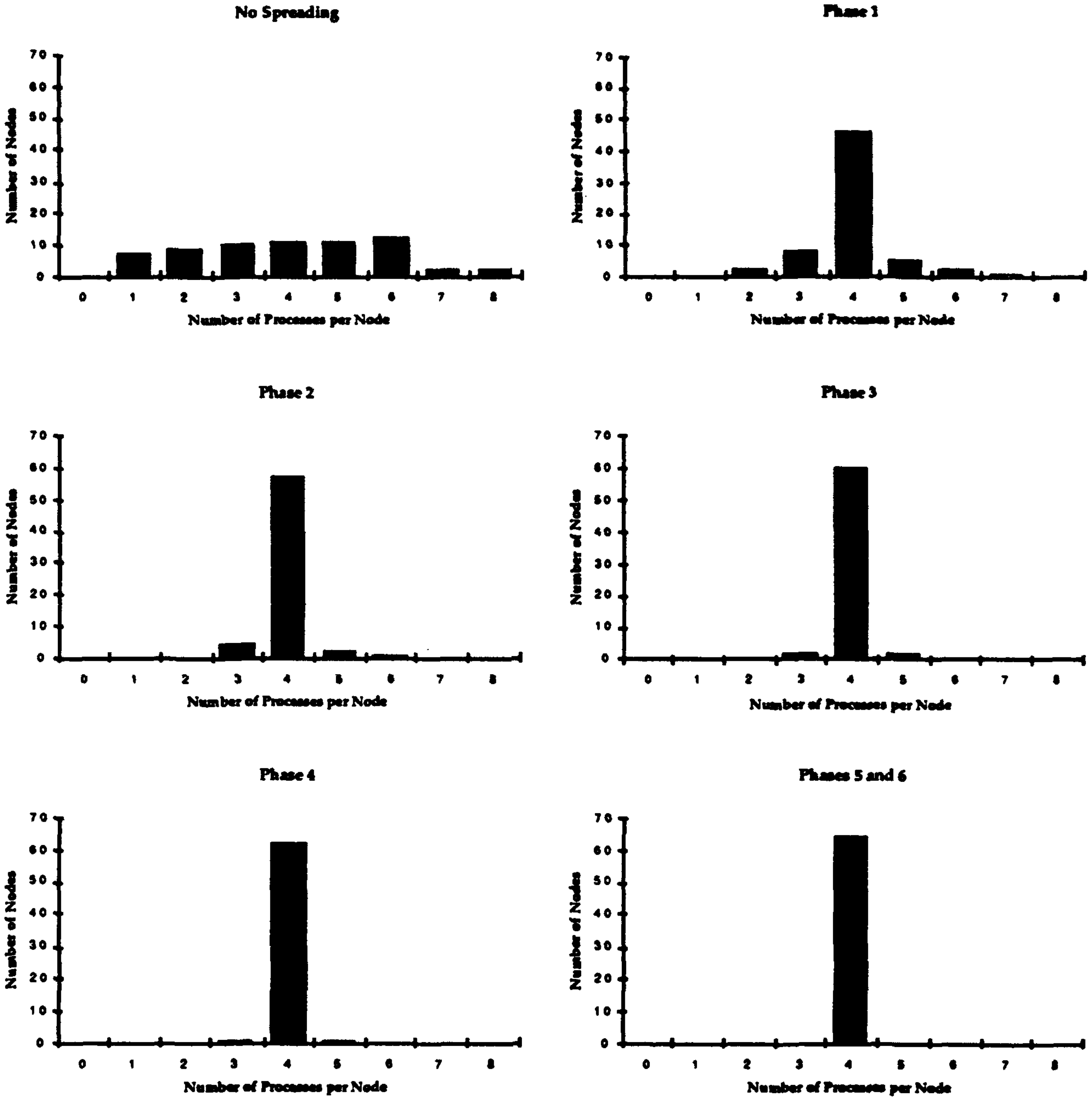


Fig. 7. Histograms for the number of processes mapped onto each node for the case of mapping 256 processes onto 64 nodes with $E[|\mathcal{W}|] = 256$.

TABLE IV. HYPERSPHERE MAPPER RESULTS: 256 PROCESSES ONTO 64 NODES

spreading	$E[\|\mathcal{W}\|] = 128$		$E[\|\mathcal{W}\|] = 256$		$E[\|\mathcal{W}\|] = 512$		$E[\|\mathcal{W}\|] = 1024$	
	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$
none	0.619	3.06	0.852	4.10	1.340	4.16	1.763	7.73
phase 1	0.850	0.45	1.000	0.87	1.433	1.28	1.850	4.05
phase 6	0.973	0.00	1.168	0.00	1.598	0.00	2.110	0.00

VIII. CONCLUSIONS

A. *Summary*

The hypersphere mapper method described in this report addresses the mapping and allocation problems that arise during parallel software development. This method is a tool that maps processes to nodes for the parallel programmer. This mapping reduces the contention for communication resources, in addition to balancing the load on each processor. This algorithm relieves the programmer from having to map the tasks and balance the load. The algorithm does this by knowing the geometry of the physical system and the communication patterns of the processes. This means it is beneficial in the testing and implementation phases of software development. It also provides a transparent programming environment to the parallel programmer, because they do not have to know the interconnection network of the physical architecture.

The hypersphere mapper approach is novel in that it utilizes techniques from nonlinear programming as a means of solving the mapping problem. In contrast, competing mapping algorithms are based on combinatoric heuristics. An important advantage of hypersphere mapper over existing mapping heuristics is that the case of mapping more than N processes onto N nodes is solved directly. These competing heuristics are limited to problems where the number of processes is no larger than the number of nodes.

The variance of the number of processes mapped to each node is introduced as a means of quantifying the measure of "computational load balance" associated with a particular mapping. The average distance between mapped communicating processes is used to measure the communication load requirement on the interconnection network. Experiences with existing hypercube multiprocessors have indicated that latencies caused by contention for the communication resources are often more of a throughput bottleneck than the uniformity of the computational load balance. Hypersphere mapper offers a means of striking a practical balance between these two competing factors.

B. Ongoing and Future Work

The hypersphere mapper could also be used for 2-D mesh systems such as the Intel Delta or Paragon by reformulating the objective function. In the 2-D case the mesh of processors could be wrapped around the n-dimensional hypershere like a blanket around a globe. The processes continuous location would then be discretized and the processes would be assigned to a processor in that region of the hypersphere.

Work is currently underway in evaluating a variety of different objective functions. By slightly altering the proposed objective function for hypersphere mapper, the resultant mappings can be tuned to have desired characteristics. For instance, the nature of the resultant mappings can be changed by simply multiplying the average Euclidean distance component of the objective function (refer to Equation $V.4$) by a scalar weighting factor, say γ. In particular, the mappings produced with $\gamma > 1$ tend to have smaller values of $\hat{f}_H(z)$ and larger values of $\hat{\sigma}^2_{\text{pmn}}$ (with no spreading), than the mappings produced with $\gamma \leq 1$.

For cases where P^2 is large relative to $|\mathcal{W}|$, the dominant computational time associated with each of the gradient projection iterations comes in computing the component of the gradient associated with the average inverted Euclidean distance between *all pairs* of processes (refer to Equations $V.4$ and $V.11$). Instead of averaging the inverted Euclidean distance over all pairs of processes, an average could be taken over a smaller set of pairs of processes. The set W appears to be a natural candidate. Another possibility is to average over a randomly generated collection of process pairs. Some preliminary studies along these lines are currently underway.

Work is currently underway in integrating the hypersphere mapper into a software engineering tool for high-performance computing systems. The tool, called the Optimized Mapping Alternate Routing System (OMARS), is a joint in-house effort by Rome Laboratory and Purdue University to develop a usable tool for optimal process allocation in massively parallel systems. The OMARS tool provides a framework for evaluating different combinations of mapping and alternate routing for process communication and would be used in the implementation and testing phases of the software development cycle. It provides a visual depiction of the logical communication of the processes and a graph of the physical system illustrating the process allocations and the link utilization. OMARS provides the users with several mapping algorithms (one of which is the hypersphere mapper) and routing algorithms. By using OMARS the user can choose an optimal mapping based on the hypersphere mapper. In addition to the default routing, an alternate routing can be calculated based on the mapping. This alternate routing pattern is considered, because the speedup and efficiency gained through the optimal mapping of the hypersphere could be lost by using the default routing found in the system. The OMARS tool combines optimal process allocation with alternate routing to increase performance.

The hypersphere mapper algorithm is currently applied during the implementation and testing phases, however, it could be applied as early as the design phase of the software engineering lifecycle. The hypersphere mapper is based on the geometry of the physical system and the communication patterns of the processes. Since the communication between functions is specified during the design, the hypersphere mapper algorithm could be applied as early as the design phase. Thus, performance analysis could begin as the parallel software is being designed and developed.

References

[BaS79] M. S. Bazaraa and C. M. Shetty, *Nonlinear Programming: Theory and Applications*, John Wiley & Sons, NY, NY, 1979.

[Bha80] K. Bhat, "On the complexity of testing a graph for N-cube," *Information Processing Letters*, **11**, pp. 16–19, 1980.

[BMS88] S. Bettayeb, Z. Miller, and I. Sudborough, "Embedding grids into hypercubes," *VLSI Algorithms and Architectures: 3rd Aegean Workshop on Computing*, Lecture Notes in *Computer Science*, Springer Verlag, **319**, pp. 201–211, 1988.

[ChG88] W.-K. Chen and E. Gehringer, "A graph oriented mapping strategy for a hypercube," *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 200–209, 1988.

[CKV87] G. Cybenko, D Krumme, and K. Venkataraman, "Fixed hypercube embedding," *Information Processing Letters*, **25**, pp. 35–39, 1987.

[CSG89] W.-K. Chen, M. F. M. Stallman, and E. F. Gehringer, "Hypercube embedding heuristics: an evaluation,"*International Journal of Parallel Programming*, Vol. 18, No. 6, pp. 505–549, 1989.

[Fox88] G. Fox, "A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube," *The IMA Volumes in Mathematics and its Applications, Volume 13*, Martin Schults, editor, Springer Verlag, 1988.

[KGV83] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, pp. 671–680, 1983.

[Wag89] A. Wagner, "Embedding arbitrary binary trees in a hypercube," *Journal of Parallel and Distributed Computing*, **7**, pp. 503–520, 1989.

[Wu85] A. Wu, "Embedding of tree networks into hypercubes," *International Journal of Parallel and Distributed Computing*, **2**, pp. 238–249, 1985.